SOFTWARE REVIEWS

Software reviews are a "filter" for the software engineering process. That is, reviews are applied at various points during software development and serve to uncover errors and defects that can then be removed. Software reviews "purify" the software engineering activities that we have called *analysis*, *design*, and *coding*.

Many different types of reviews can be conducted as part of software engineering. Each has its place. An informal meeting around the coffee machine is a form of review, if technical problems are discussed. A formal presentation of software design to an audience of customers, management, and technical staff is also a form of review.

- **FORMAL TECHNICAL REVIEWS (FTR)**

A formal technical review is a software quality assurance activity performed by software engineers (and others). The objectives of the FTR are

(1) to uncover errors in function, logic, or implementation for any representation of the software;

(2) to verify that the software under review meets its requirements;

(3) to ensure that the software has been represented according to predefined standards;

(4) to achieve software that is developed in a uniform manner; and

(5) to make projects more manageable.

- **INFORMAL TECHNICAL REVIEWS (ITR)**

ITRs allows more casual approach in which one or more reviewers help the software engineer improve his work products. Typically, these meeting do not follow any particular plan and no formal minutes are recorded. pair programming is example on the review oriented aspects.

REVIEW METRICS

Technical reviews are one of many actions that are required as part of good software engineering practice. Each action requires dedicated human effort.

Before analysis can begin, a few simple computations must occur. The total review effort and the total number of errors discovered are defined as:

- $E_{review} = E_p + E_a + E_r$

Where Preparation effort, E_p—the effort (in person-hours) required to review a work product prior to the actual review meeting, and Assessment effort, E_a— the effort (in person-hours) that is expended during the actual review, and Rework effort, E_r — the effort (in person-hours) that is dedicated to the correction of those errors uncovered during the review.

- $\mathrm{Err}_{tot} = \mathrm{Err}_{minor} + \mathrm{Err}_{major}$

Where Err_{minor} represent Minor errors found,—the number of errors found that can be categorized as minor (requiring less than some prespecified effort to correct) , and Err_{major} represent Major errors found,—the number of errors found that can be categorized as major (requiring more than some prespecified effort to correct).

Error density represents the errors found per unit of work product reviewed.

$$\text{Error density} = \frac{\text{Err}_{\text{tot}}}{\text{WPS}}$$

-

Where Work product size, WPS—a measure of the size of the work product that has been reviewed (e.g., the number of UML models, or the number of document pages, or the number of lines of code)

For example, if a requirements model is reviewed to uncover errors, inconsistencies, and omissions, it would be possible to compute the error density in a number of different ways. The requirements model contains 18 UML diagrams as part of 32 overall pages of descriptive materials. The review uncovers 18 minor errors and 4 major errors.

Therefore, Err_{tot} 22. Error density is 1.2 errors per UML diagram or 0.68 errors per requirements model page.

If reviews are conducted for a number of different types of work products (e.g., requirements model, design model, code, test cases), the percentage of errors uncovered for each review can be computed against the total number of errors found for all reviews. In addition, the error density for each work product can be computed.

Once data are collected for many reviews conducted across many projects, average values for error density enable you to estimate the number of errors to be found in a new (as yet unreviewed document). For example, if the average error density for a requirements model is 0.6 errors per page, and a new requirement model is 32 pages long, a rough estimate suggests that your software team will find about 19 or 20 errors during the review of the document. If you find only 6 errors, you’ve done an extremely good job in developing the requirements model or your review approach was not thorough enough.

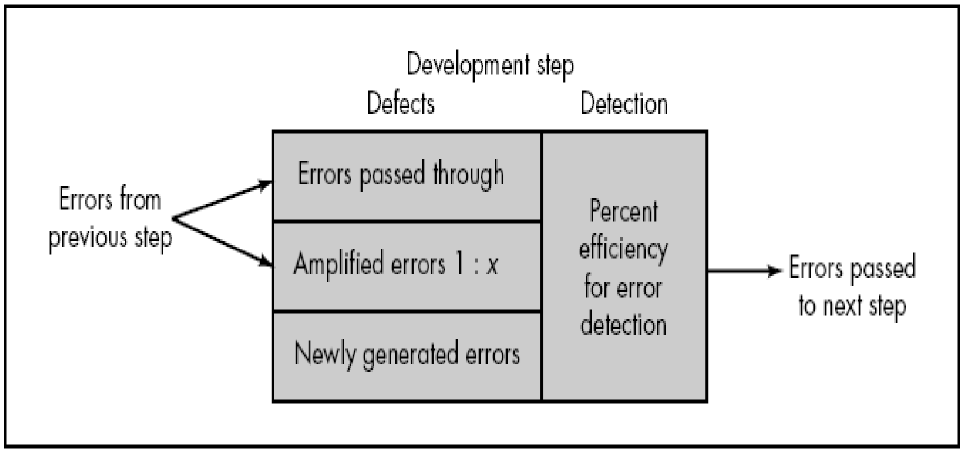
Once testing has been conducted, it is possible to collect additional error data, including the effort required to find and correct errors uncovered during testing and the error density of the software. The costs associated with finding and correcting an error during testing can be compared to those for reviews.

Defect Amplification and Removal:

A defect amplification model can be used to illustrate the **generation and detection of errors** during the preliminary design, detail design, and coding steps of the software engineering process. The model is illustrated schematically in Figure below. A box represents a software development step. During the step, errors may be inadvertently generated. Review may fail to uncover newly generated errors and errors from previous steps, resulting in some number of errors that are passed through.

In some cases, errors passed through from previous steps are amplified (amplification factor, *x*) by current work. The box subdivisions represent each of these characteristics and the percent of efficiency for detecting errors, a function of the thoroughness of the review.

FIGURE 8.2
Defect amplification model



First Figure below illustrates a hypothetical example of defect amplification for a software development process in which no reviews are conducted. Referring to the figure, each test step is assumed to uncover and correct 50 percent of all incoming errors without introducing any new errors. Ten preliminary design defects are amplified to 94 errors before testing commences. Twelve latent errors are released to the field. Second Figure considers the same conditions except that design and code reviews are conducted as part of each development step. In this case, ten initial preliminary design errors are amplified to 24 errors before testing commences and only three latent errors exist.

To conduct reviews, you must expend time and effort, and your development organization must spend money. However, the results of the preceding example leave little doubt that you can pay now or pay much more later.

FIGURE 8.3
Defect amplification, no reviews

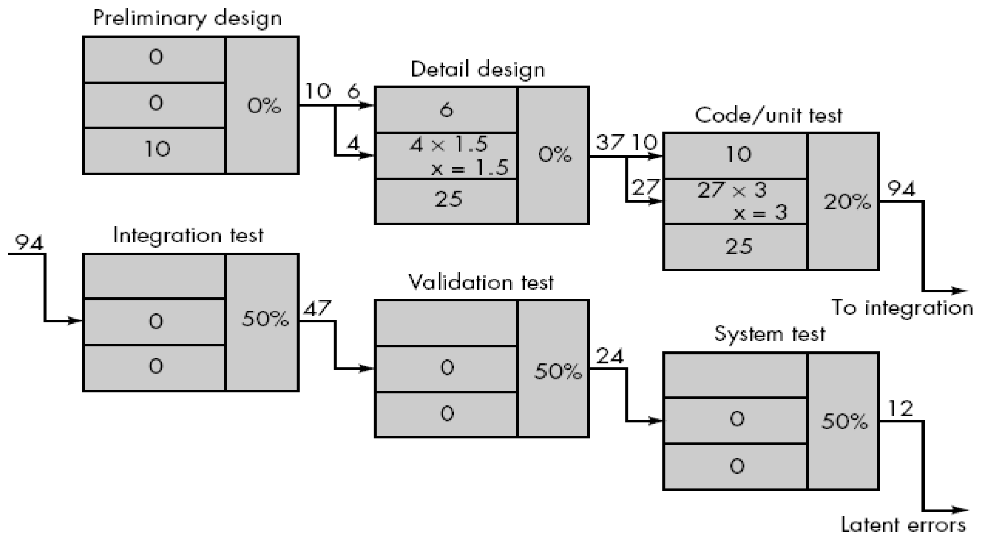
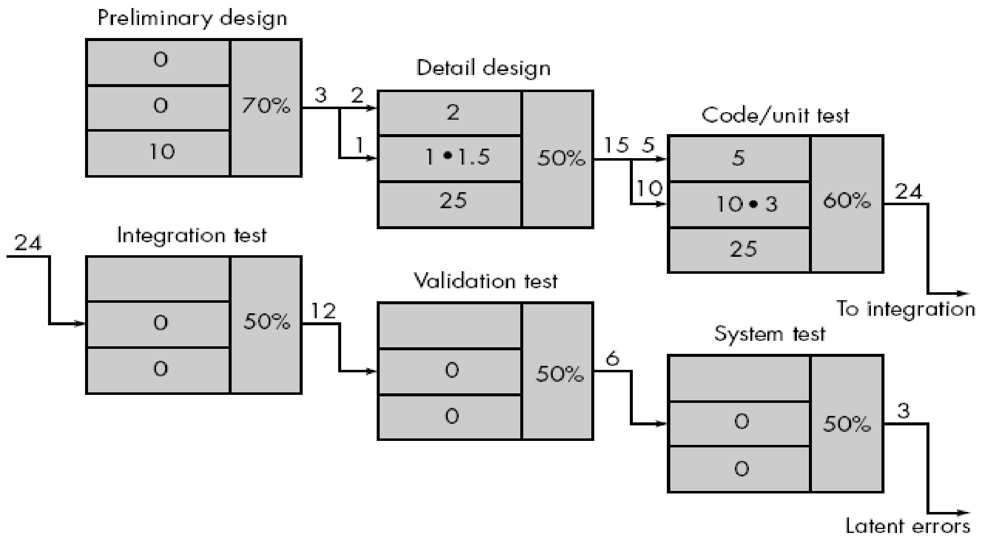


FIGURE 8.4
Defect amplification, reviews conducted


Preliminary design
0
0
10
70%
3
2
1
Detail design
2
1 • 1.5
25
50%
15
5
10
Code/unit test
5
10 • 3
25
60%
24
To integration
24
Integration test
0
0
50%
12
Validation test
0
0
50%
6
System test
0
0
50%
3
Latent errors